A New Approach in Key Generation and Expansion in Rijndael Algorithm

Naim Ajlouni[1], Asim El-Sheikh[2], and Abdullah Abdali Rashed[2]
[1]Amman Arab University for Graduate Studies, Jordan
[2]Arab Academy for Banking and Financial Science, Jordan

Abstract: *This paper presents a new algorithm that simplifies the process of generating and expanding cipher key, which is considered one of the most important elements in ciphering process. The algorithm generates a random pool of keys (long size key), this key is sent to the authorized receiver. During the ciphering process the algorithm will select the schedule keys randomly from the pool of keys. The receiver will be given the index of the first element in the schedule key and the key length. During the deciphering process the deciphering algorithm will use the received information to extract the schedule from the original pool, this key is then used to decipher the ciphered data block without any key re-expansion.*

Keywords: *Rijndael block cipher, AES, key generation, key expansion.*

Received September 1, 2004; accepted December 2,12004

1. Introduction

Rijndael is a cipher algorithm with a simple and elegant structure [5] as it is based only on the most simple imaginable operations [2]. However it is one of the most important algorithms in modern cryptography [7] as it is a block cipher that provides a mapping from plaintext blocks to cipher text blocks and vice versa using the same ciphering key [6].

The National Institute of Standards and Technology (NIST) has selected Rijndael in October 2000 to replace DES and adopted it formally in December 2001 [4, 8] to be used by U.S. government organizations (and others) to protect sensitive information [12].

Encryption and decryption generally require the use of some secret information, referred to as a ciphering key. This ciphering key must be kept secret so that unauthorized parties cannot, even with knowledge of the algorithm, complete the decryption process [15]. In Rijndael algorithm, the same key is used for both encryption and decryption [11]; in other mechanisms, the keys used for encryption and decryption is different [13].

If the ciphering key is not random and independent, the ciphered data may become weak to related-key attacks, and for that reason developers should choose how to generate subkeys [1]. It is useful to avoid using the conventional key scheduling process, and specify the ciphering keys which are both random and independent directly in the ciphering key [1].

Ciphering key is a very important part of Rijndael algorithm as it will be expanded to obtain schedule key which is used in add round key phase. The receiver or decryptor should know the ciphering key or schedule key [11].

Although Rijndael is more than likely secure enough for all applications in the real world [14], in this paper, it is proposed to present a new approach of ciphering key generation that will make it more suitable and amenable for real world applications.

2. Rijndael Algorithm Specification

Ciphering key is expanded to schedule key [3] that is used (partitioned to round keys) for encryption, which is converting plaintext to an unintelligible form called ciphertext; decrypting the ciphertext however, converts cipheredtext back into its original form, called plaintext [9]. Rijndael algorithm has three different combinations according to Nk that might be 4, 6 or 8 words [10] as shown in Table 1.

Table 1. Ciphering key-block combinations [9].

	Key Length (N_k words)	**Block Length (N_b words)**	**Number of Rounds N_r**
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

The Four fundamental transformations [10, 11, 16] of Rijndael algorithm are as follows:

- *SubBytes*: In this part the elements of the input are substituted by values from sBox matrix.
- *The shiftrows*: In this part the rows of the state are cyclically shifted over different offsets. Row 0 is not shifted; row 1 is shifted over 1 byte, row 2 over 2 bytes and row 3 over 3 bytes.
- *MixColumn*: In this part the columns of the state are considered as polynomials over *GF* (2^8) and

multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by:

$$C(x) = '03' x^3 + '01' x^2 + '01' x + '02'$$

- *Key Addition*: In this part the round key is applied to the State by simple bit by bit XOR.

3. Static Ciphering Keys with Random Alternatives

Ciphering key is used to generate the schedule key that is actually used in turn to cipher the plaintext or decrypt the ciphered-text. This method is based on generating 0...2047 random numbers, as shown in Table 2, the random numbers are to be considered as a large fixed keys or "*pool of keys*" (stage 1), this is sent to the receiver, the schedule key will be selected from this pool of keys as in the following algorithm:

Table 2. Proposed key elements.

Index no.	0	1	2	…	2046	2047
Random no.						

During ciphering the schedule key is selected from the large fixed keys "pool of keys" at random, this is the starting point of (stage 2). This operation will be done for every cipher phase and key length plus the starting point will be sent to the receiver every time.

3.1. Generating Fixed Key

```
Function generating the fixedKeys.
Output: A matrix of 2048 elements.
  Begin
    For i = 0 to 2047 step by 1
      $fixedKeys_i$ = Generate a random number
    end for
    return the generated numbers
End Function
```

3.2. Schedule Key Algorithm

```
Function generate schedule key from fixed matrix
Input:  chedule key length
Output: schedule key
  Begin
    Let startPoint = random number from 1 to [2048-schedule key length]
    Return matrix from 0 to schedule key length
End function
```

3.3. Cipher Algorithm

The ciphering algorithm is as shown below:

```
Function doCipher
Input: Nk, plain file
Output ciphered file
Begin
  Open ciphered file for writing
  Open plain file for reading
  While it is not the end of the file do
      Read 16 bytes from the input file
      Convert the 16 bytes to hexadecimal format
      Call schedule key from fixed matrix
      Ciphered data block = cipher (Nk, input, schedule key)
      Write Ciphered data block to the output file
  End while
End function decipher
```

3.4. Deciphers Algorithm

```
Function doDecipher
Input: ciphered file, schedule key start point, Nk
Output: plain file
  Begin
    Open ciphered file for reading
    Open plain file for writing
      While it is not the end of the file
        Read 32 byte as ciphered data block
        Schedule key = FixedKeys [start point ...schedule key length-1]
        Plain text = decipher (Nk, schedule key, ciphered data block, schedule key)
        Convert plain text from hexadecimal to string
        Write the plain block to the plain file
      End while
End function
```

4. Example

Assuming that in this example:

$$Nk = 4, Nb = 4, Nr = 10$$

Suppose that the fixed key is generated as in Table 3. The first random generated number, Start row = 6 that means row 6. Whereas the second random generated number Start column = 7 that means column 7.

The starting point would be (7, 6) which is generated randomly whereas the last point would be at (50, 10) that was calculated as shown in the following calculations:

- Last point row = start of the $Row + Nb * (Nr + 1) = 7 + 4 * (10 + 1) = 6 + 44 = 50$.
- Last point column = start of the $column + Nb - 1 = 7 + 4 = 6 + 4 = 10$.

The starting point of the schedule key is (7, 6) to (50, 10).

The sender would just send the ciphered data block and key length plus the starting point of the schedule key that he or she used in encrypt the text.

Therefore, the schedule key would be as in Figure 1.

Table 3. Pool of fixed key generated for once randomly.

b3	9d	c0	d6	ae	a9	1f	1e	f2	de	db	0e	bc	3d	06	2f
8b	4b	c3	2f	b4	13	8b	98	d3	4d	7b	c5	1a	25	11	e4
62	6f	a7	d3	8c	bc	54	39	eb	29	9c	cc	91	21	d1	1d
cc	df	ec	eb	7c	96	3a	08	cf	b0	16	33	ee	e6	62	28
c1	76	fd	3^{a}	35	8f	ad	09	f3	e9	bc	3d	4d	61	94	5e
57	aa	d2	dc	b1	c9	**8e**	**4a**	**8f**	**77**	86	78	c3	8b	0a	8b
38	c4	1c	f6	b6	ad	**ab**	**10**	**85**	**22**	62	13	6e	ee	4a	d4
97	65	25	1e	51	24	**cd**	**35**	**eb**	**ba**	12	d2	64	3f	06	9e
98	3e	1c	73	d5	dd	**da**	**ec**	**0d**	**61**	bb	79	20	42	63	37
07	63	7a	d2	4a	f5	**cf**	**15**	**56**	**b2**	22	60	82	6f	a0	f5
09	98	96	dc	1a	d9	**71**	**a7**	**07**	**d2**	1c	87	ed	2d	07	76
df	20	9e	20	c4	f0	**be**	**77**	**ad**	**ad**	6c	b2	94	e5	1c	40
3b	59	be	9c	a3	e3	**b5**	**8b**	**f5**	**f8**	13	9a	7d	c9	ae	f8
4a	8f	54	48	45	b0	**1a**	**41**	**56**	**ab**	8b	1a	e9	dd	70	1d
26	17	3a	dc	83	20	**d3**	**79**	**76**	**d4**	88	2d	08	58	7b	a8
81	df	a6	64	92	83	**d5**	**ba**	**42**	**ff**	c6	8b	c6	a2	fc	ea
6a	2b	8a	17	62	21	**b4**	**45**	**3f**	**00**	78	e0	16	c3	f2	eb
af	d6	72	c2	9a	41	**43**	**7f**	**1a**	**b1**	16	7c	27	00	ed	44
cd	be	60	87	ab	64	**4e**	**99**	**d5**	**92**	0b	a1	8c	cc	4e	98
9c	74	ef	aa	69	86	**1e**	**63**	**d3**	**3a**	34	53	10	02	61	85
af	ea	5b	c3	90	e8	**42**	**e2**	**0d**	**2d**	a0	34	9f	0c	01	00
32	13	18	4^{a}	8a	94	**51**	**ca**	**1b**	**76**	6a	e7	59	05	c0	37
b8	ab	79	16	f6	77	**59**	**74**	**82**	**ab**	ff	d9	d1	c5	4e	fa
6a	a3	41	19	82	63	**9c**	**41**	**08**	**d5**	d3	c4	75	94	41	a1
15	de	64	26	7a	6b	**2d**	**83**	**4d**	**c4**	a8	b9	34	51	b3	a3
Dc	20	a1	55	45	a3	**a1**	**d6**	**6d**	**5a**	47	24	09	cb	26	3b
6f	30	46	2c	69	fc	**fe**	**1f**	**fb**	**7b**	d3	46	fa	93	7e	cf
21	2e	a9	74	e3	b7	**c7**	**bd**	**54**	**3a**	33	e5	65	c6	91	39
a3	4f	e9	ef	ea	df	**3b**	**e4**	**8a**	**4c**	a2	0b	d2	73	26	98
02	9b	be	e1	a9	8b	**96**	**9e**	**b4**	**a0**	bf	96	ae	41	01	ee
28	8b	a3	c6	c5	b8	**1c**	**cb**	**fb**	**41**	8c	cf	cb	75	15	b0
d7	ad	4b	03	21	df	**3b**	**77**	**da**	**c6**	c9	46	0f	09	b0	9e
dc	56	78	76	97	8a	**a6**	**28**	**2e**	**fe**	15	50	22	02	cc	3d
18	12	02	45	31	0a	**f7**	**aa**	**90**	**cc**	eb	d7	5b	69	f9	79
d5	d1	cd	68	86	52	**f9**	**7e**	**38**	**7f**	04	b8	c3	6d	80	ad
0e	d4	9c	4c	dc	39	**ab**	**cc**	**58**	**d4**	61	cd	6d	ce	10	ad
f1	1f	b2	6f	b7	71	**ce**	**dc**	**2e**	**40**	7d	17	9a	b0	0c	17
ff	44	b9	62	33	2f	**80**	**d9**	**2f**	**04**	38	2d	c4	04	6b	6b
27	14	e7	01	e7	71	**fd**	**5b**	**18**	**56**	8a	ea	2f	ae	72	14
04	3c	3c	5b	e3	35	**1e**	**3f**	**7a**	**94**	0c	dd	17	58	59	Cd
02	a6	9b	04	79	dd	**27**	**e5**	**3e**	**5f**	ac	76	2d	00	a5	87
45	b1	01	80	85	99	**64**	**92**	**5b**	**9f**	49	36	96	16	39	Ef
69	6a	9b	0a	a3	9d	**d9**	**4e**	**bb**	**28**	71	8a	da	3a	c2	f8
c7	5f	fb	b5	21	13	**c1**	**fc**	**6a**	**43**	0e	51	74	fb	77	Bb
aa	08	31	f9	6e	97	**7b**	**f5**	**ce**	**65**	ba	df	ac	95	98	26
73	fc	30	64	e9	2c	**fa**	**c9**	**cb**	**c3**	f7	09	dd	33	6c	49
98	d8	87	43	6d	30	**61**	**69**	**c3**	**09**	63	67	7a	26	6e	86
10	b6	d1	6c	3f	16	**05**	**9b**	**2d**	**23**	a2	9c	84	0c	7d	a0
52	33	f4	3d	02	da	**b9**	**d3**	**fc**	**90**	c1	3d	76	d7	0d	f6
74	66	11	47	af	a8	9e	98	b6	9b	3e	28	73	22	1f	a6
7a	45	ae	27	78	e5	9f	6b	ca	20	3f	97	44	5a	b7	b7
56	dd	1b	c7	24	61	72	2c	48	35	89	5c	19	92	a3	4e
1f	19	5a	56	7d	51	06	03	93	1d	43	e4	d6	3b	30	93
3c	75	51	b2	c3	a4	02	e4	2e	30	42	9b	0e	a9	a6	5b
85	ab	29	6^{a}	c5	19	c2	6a	eb	83	da	cb	7b	cc	05	ca
ac	f2	2b	d8	f2	49	e4	06	2c	Ac	98	e8	90	e4	94	be
b7	a9	0e	75	97	29	ac	4e	2c	Fb	e8	cf	a3	29	d4	f2
0e	8f	16	e0	20	e2	2e	da	c1	be	8b	d7	f3	40	d7	37
da	c3	46	38	01	0a	5e	f6	0d	59	9b	eb	9c	45	04	58
c5	91	a8	ca	3a	c4	4c	3e	e7	7d	c9	df	23	0b	e4	94
b9	97	00	e3	d4	11	59	d9	fb	2c	5a	25	84	fb	8f	81
3a	f8	8a	6c	1c	ec	1d	12	0e	02	a4	c7	22	b2	4f	83
58	87	a8	37	96	25	fc	19	a4	7c	d2	ab	25	bf	97	a6
60	eb	7e	83	3b	fa	d2	0d	24	9c	5c	49	aa	12	88	df
a4	48	32	55	5e	3d	2d	42	f3	ec	eb	77	ef	17	09	fe
65	19	37	a9	71	40	4e	5e	17	e0	61	d2	3e	dd	3e	78
eb	90	62	ad	6b	87	4b	10	db	4d	81	62	84	bf	9f	f2
a3	e2	9e	e6	43	7f	4e	73	dc	d9	98	09	24	8d	ce	22
1e	ca	fd	87	ec	89	ba	ed	60	35	c1	39	78	a2	a7	fa
ab	03	cf	21	d2	79	88	54	14	c1	05	7b	e8	56	00	d6
58	2d	d9	5a	3b	8f	46	c7	8c	14	ab	50	41	b9	27	bb
52	14	5f	86	a2	6f	56	fe	d2	d3	e4	9e	b7	15	91	54
b1	06	91	3c	8a	65	fb	ea	81	27	bb	85	25	e7	bc	83
f5	7a	6f	5d	e8	fe	99	d5	d8	93	55	d0	08	17	74	fe

ca	d7	e0	19	04	5b	c4	87	e6	6b	0a	14	f0	3c	de	ac
a1	18	29	69	27	1b	cb	45	b6	bb	7b	90	5d	53	ca	f8
b2	46	03	b0	fc	f0	0b	1c	76	9d	ec	b7	1e	a4	be	ea
18	7c	51	99	78	66	1b	f4	ba	4e	42	f6	31	3f	27	fa
93	0f	2b	84	7e	69	0a	fa	64	27	32	67	90	68	56	cb
c2	b4	d9	37	6f	ba	50	40	1e	53	b4	3e	6e	d7	20	cd
04	e4	49	cc	c1	2a	5a	bf	18	54	6a	a9	09	aa	79	a8
97	b8	f8	da	f5	9b	f9	40	bb	7b	b5	5b	a5	ac	eb	6e
26	fa	33	90	e1	3b	83	ee	41	1e	3d	4f	be	8c	95	75
76	c0	63	23	b2	97	1c	62	69	6d	49	44	e4	e2	af	fc
07	4d	bf	ae	16	f4	5c	68	e2	1b	d2	ab	89	79	f0	e0
a1	0e	f8	50	80	36	57	d0	8d	06	0a	e6	32	41	d5	17
41	1d	28	b7	60	33	ce	ee	55	d5	fa	59	d4	ca	a9	72
7c	15	f0	c8	08	dc	1d	59	98	9b	f9	e4	b2	16	58	45
29	ae	d9	00	f7	33	dd	64	3a	82	b4	76	be	e3	69	4e
41	31	1e	f1	06	d4	7c	26	9b	99	a6	f7	a3	01	32	8c
90	7c	ef	d1	cc	b5	c0	ce	45	16	bb	8b	e7	45	d2	13
ee	5e	1e	fa	15	85	e5	59	85	a2	da	ac	5b	96	b1	d5
33	26	5c	06	e9	db	db	3d	d1	48	f7	6f	95	7c	3f	a8
13	e3	49	fe	03	10	d5	3c	02	b1	3f	68	30	be	63	9b
6b	60	e1	5d	62	17	74	d8	51	8a	ea	55	cd	02	a6	10
e4	25	4f	66	4b	b0	1c	11	ab	17	36	a3	ef	d6	37	3e
0d	f5	25	3b	53	81	5c	e3	9a	ff	96	01	c9	a9	f5	8b
6b	c7	df	f3	be	8a	33	74	98	1d	83	8c	f0	1e	2f	8f
b9	e1	9e	f4	9f	8f	38	d4	89	aa	47	a1	d6	14	c3	af
05	38	5d	d4	b9	7b	6d	b9	1f	73	7e	57	59	0c	9b	70
dd	4b	67	a3	36	1b	da	10	a1	2c	d4	92	93	0f	70	ef
40	75	3b	32	1c	8f	70	bb	e2	2d	a3	1e	b7	e8	31	38
4e	39	ba	07	1c	63	53	10	8a	04	93	c0	51	64	27	5a
d2	15	54	b3	3a	b7	db	d1	c5	5b	55	b1	34	6f	d4	10
b3	0a	67	cf	41	33	d4	98	22	52	36	82	06	07	bc	f7
73	61	fe	38	c2	b0	b6	48	24	71	e8	34	e4	d2	ba	c3
8b	42	3e	30	9d	cd	ad	33	cd	13	f9	9e	2a	f8	eb	58
4d	5c	88	de	37	53	be	85	4e	3a	97	43	e6	27	2c	0f
81	58	b3	94	f2	92	2e	e8	e5	e6	57	f8	d6	17	0a	bb
61	35	ed	0c	3d	40	39	e8	3c	7b	bf	9d	a5	96	e1	e4
8c	19	3a	c6	47	c7	e6	60	59	47	96	41	d9	9b	49	18
22	5b	41	86	08	43	52	48	e8	6b	52	c9	64	4d	e4	b5
b1	28	fb	75	71	d2	a3	c4	3f	51	28	67	72	8a	a0	40
38	59	13	4b	6c	c1	9f	e5	d8	7e	b8	51	c5	bd	41	c9
2f	9b	f1	35	4a	9b	32	20	02	0e	0d	a6	a5	26	d2	a1
1c	a6	3c	87	80	4a	69	f7	80	5a	dc	96	60	0c	93	f6
6f	60	27	4b	87	7e	91	be	63	f9	90	b8	07	ab	59	72
cf	28	bf	c0	19	57	84	f0	41	ee	dc	f0	fc	c3	7a	f8
ea	b8	1c	a3	23	28	db	98	5a	9a	d8	c4	a2	64	5c	76
33	80	2b	f7	50	3d	aa	18	fc	42	9e	89	49	40	05	1c
82	e8	51	4a	d6	06	4d	5c	32	19	0a	50	50	c6	e7	58
af	0d	47	1b	dc	58	eb	13	9f	03	18	74	51	45	64	83
05	5e	4a	b2	5e	89	41	35	9e	5d	04	5d	70	23	a3	60
7d	7c	37	84	c6	c1	31	7b	94	a4	37	c8	3a	17	29	ee
bd	99	fc	4f	82	f5	51	e1	58	fb	e5	2c	f3	09	4f	a0
2a	55	56	f8	b8	eb	ea	03	76	11	cd	62	f5	90	18	e3
71	30	7f	39	03	af	36	ed	9c	79	85	6a	8d	01	f7	e4
6a	b0	8d	56	de	af	3f	67	7a	8e	9d	14	fd	5d	35	89

8e	4a	8f	77
ab	10	85	22
cd	35	eb	ba
da	ec	0d	61

cf	15	56	b2
71	a7	07	d2
be	77	ad	ad
b5	8b	f5	f8

1a	41	56	ab
d3	79	76	d4
d5	ba	42	ff
b4	45	3f	00

43	7f	1a	b1
4e	99	d5	92
1e	63	d3	3a
42	e2	0d	2d

51	ca	1b	76
59	74	82	ab
9c	41	08	d5
2d	83	4d	c4

a1	d6	6d	5a
fe	1f	fb	7b
c7	bd	54	3a
3b	e4	8a	4c

96	9e	b4	a0
1c	cb	fb	41
3b	77	da	c6
a6	28	2e	fe

f7	aa	90	cc
f9	7e	38	7f
ab	cc	58	d4
ce	dc	2e	40

80	d9	2f	04
fd	5b	18	56
1e	3f	7a	94
27	e5	3e	5f

64	92	5b	9f
d9	4e	bb	28
c1	fc	6a	43
7b	f5	ce	65

fa	c9	cb	c3
61	69	c3	09
05	9b	2d	23
b9	d3	fc	90

Figure 1. Schedule key taken from the large fixed key Nk = 4, Nb = 4, Nr = 10

5. Illustrative Example: Vector Cipher Examples

All vectors are represented in hexadecimal format as a hexadecimal number can be represented as one byte as it consists of two digits (4 bits four one hexadecimal digit).

The phrases used in this case are the standard AES has been used. Date: Thu Jul 01 10:36:00 IDT 2004.

5.1. Cipher Example: 128-bit Ciphering Key

Figure 2 shows the hexadecimal values in the state array as the ciphering processes for an input block length 16 bytes and ciphering key length 4 words.

- *Input String* = 41 73 696d 20 41 20 45 6c 2d 53 68 65 69 6b 68.
- *Ciphering key* = 8e ab cd da 4a 10 35 ec 8f 85 eb 0d 77 22 ba 61.

Round No.	Start of Round				After SubByte				After Shiftrow				After Mixcolumns				Round Key Values				
Input	41	20	6c	65													8e	4a	8f	77	
	73	41	2d	69													ab	10	85	22	=
	69	20	53	6b													cd	35	eb	ba	
	6d	45	68	68													da	ec	0d	61	
1	cf	6a	e3	12	a8	02	11	C9	a8	02	11	c9	a0	ce	76	3e	cf	15	56	b2	
	d8	51	a8	4b	61	d1	c2	B3	d1	c2	B3	61	86	76	64	ad	71	a7	07	d2	=
	a4	15	b8	d1	49	59	c6	3e	c6	3e	49	59	80	c5	5e	cd	be	77	ad	ad	
	b7	a9	65	09	a9	d3	4d	01	01	a9	D3	4d	a3	b3	74	e2	b5	8b	f5	f8	
2	c5	Db	20	8c	a6	b9	b7	64	a6	b9	B7	64	ba	dc	Ad	8d	1a	41	56	ab	
	f7	d1	63	7f	68	e3	fb	d2	e3	fb	D2	68	f6	4c	c2	b0	d3	79	76	d4	
	e3	2b	f3	60	b2	f1	0d	d0	d0	d0	B2	f1	f7	6c	13	e1	d5	ba	42	ff	
	f8	38	81	1a	73	07	c0	a2	a2	73	07	0c	d9	1d	ac	2d	b4	45	3f	00	
3	a0	9d	fb	26	e0	5e	0f	f7	e0	5e	0f	f7	73	e7	1d	70	43	7f	1a	b1	
	bc	35	b4	64	65	96	d8	43	96	d8	43	65	67	c6	0c	e0	4e	99	d5	92	=
	aa	d6	51	1e	ac	f6	d1	72	d1	72	Ac	f6	bc	c3	b1	1a	1e	63	d3	3a	
	29	58	93	d2	a5	6a	dc	d8	d8	a5	6a	dc	d7	4c	2a	32	42	e2	0d	2d	
4	30	98	07	c1	04	46	c5	78	04	46	C5	78	53	e4	8f	28	51	ca	1b	76	
	29	f	d9	72	a5	e6	35	40	e6	35	40	a5	f	c4	ef	de	59	74	82	ab	=
	a2	a0	62	20	a3	e0	aa	b7	aa	b7	3a	e0	f6	78	c6	49	9c	41	08	d5	
	95	Ae	27	1f	a2	e4	cc	c0	c0	2a	E4	cc	db	1c	fd	4e	2d	83	4d	c4	
5	02	84	94	e5	77	f5	22	58	77	f5	22	58	29	66	21	ce	a1	d6	6d	5a	
	af	b0	6d	75	79	e7	3c	9d	e7	3c	9d	79	a5	1c	de	7b	fe	1f	fb	7b	=
	a6	39	ce	9c	02	12	b8	de	b8	de	02	12	f1	02	cd	37	c7	bd	54	3a	
	f6	9f	b0	8a	42	db	e7	7e	e7	42	Db	e7	09	87	54	56	3b	e4	8a	4c	
6	88	b0	4c	94	c4	e7	29	22	c4	e7	29	22	52	60	d6	8a	96	9e	b4	a0	
	a4	03	25	00	49	b7	3f	63	b7	3f	63	49	b9	d8	b7	b5	1c	cb	fb	41	=
	d8	Bf	99	0d	61	08	ee	d7	ee	d7	61	08	85	08	e9	5c	3b	77	da	c6	
	32	63	de	1a	23	fb	1d	a2	a2	23	Fb	1d	d9	9c	94	1d	a6	28	2e	fe	
7	c4	Fe	d9	2a	c1	bb	35	e5	c1	bb	35	e5	b5	4c	93	fd	f7	aa	90	cc	
	a5	13	4c	f4	06	d7	29	bf	d7	29	Bf	06	da	d8	34	70	f9	7e	38	7f	=
	be	7f	44	9a	ae	d2	1b	b8	b1	b8	Ae	d2	64	c4	41	5b	ab	cc	58	d4	
	b3	b4	ba	e3	e2	8d	f4	11	11	e2	8d	f4	60	98	f4	13	ce	dc	2e	40	
8	42	e6	03	31	c2	8e	7b	c7	c2	8e	7b	c7	d0	89	fd	20	80	d9	2f	04	
	23	a6	0c	0f	26	24	fe	76	24	fe	76	26	ee	18	09	34	fd	5b	18	56	=
	cf	08	19	8f	a8	30	d4	73	d4	73	8a	30	97	a1	2f	ab	1e	3f	7a	94	
	ae	44	61	53	e4	1b	ef	ed	ed	e4	1b	ef	45	d7	47	81	27	e5	3e	5f	

9	d8	50	d2	24	d5	53	b5	36	d5	53	B5	36	75	e4	10	56	64	92	5b	9f	=
	13	43	11	62	d7	1a	82	aa	a1	82	Aa	7d	b6	79	2b	67	d9	4e	bb	28	
	89	e9	55	3f	a7	0b	fc	75	fc	75	A7	0b	83	de	2f	9c	c1	fc	6a	43	
	62	32	79	de	aa	23	b6	1d	d1	aa	23	b6	b3	4d	8f	5b	7b	f5	ce	65	
10	11	76	b4	c9	82	38	b3	dd	82	38	B3	dd					fa	c9	cb	c3	=
	b2	37	90	4f	37	a9	60	84	a9	60	84	37					61	69	c3	09	
	42	22	45	df	c2	93	6e	9e	e6	9e	2c	93					05	9b	2d	23	
	40	b8	41	3e	09	c6	83	b2	b2	09	6c	83					b9	d3	fc	90	
output	78	fl	78	1e																	
	fb	09	47	3e																	
	b6	05	01	b0																	
	b0	da	90	13																	

Figure 2. Arrays of cipher phase.

6. InvGenerate Cipher Key 2048

There is no inverse for the previous method as sender sends only the key length plus the index of the schedule key starting point plus the key length. Using this information the ciphering key and will be extracted by decipher.

Example:

Suppose that the fixed key is generated as in Table 3. The following information should be sent to the legitimate receiver:

- 78fb6b0bf10905da784701901e3eb013 this represents the ciphered data block.
- 4 represent the length.
- Schedule key starting point (6, 5) where 6 represents row number and 5 column number. However the last point should be at (49,10) which is calculated using the following formula:

    a. Last point row = *start of the row +Nb * (Nr + 1) – 1 = 5 + 4 * (10 + 1) – 1 = 5 + 44 = 49.*

    b. Last point column = *start of the column + Nb – 1 = 6 + 4 – 1 = 6 + 44 = 10.*

The starting point of the schedule key is (6, 5) to (49, 10). So legitimate receiver will easily extract the schedule key and start decipher phase.

7. Advantages

Practically it is the fastest approach as:

- Less computations (as the key is generated only one time and used as many times) is needed. However, key expansion will not be used any more.
- In this case, instead of sending the schedule key generation (minimum 44 words at least), this algorithm will only send the two bytes, which represents key length plus the starting point.
- Many operations and methods related to schedule key are eliminated as they will not be needed.
- No need to expand the ciphering key as the proposed method uses ready schedule key.
- It is difficult to predict the numbers in the key, since this depends on random number to determine the start of the schedule key.
- The proposed method can be used in all Rijndael keys issues (AES128, AES196 AES256).
- The size of the sent data would be very small in comparison with any other method.
- No need to have an inverse class for this algorithm.

8. Disadvantages

It is not dynamic approach as the key is expanded and sent to the receiver. So, other receivers should receive the pool of keys in order to be able to decipher the received encrypted data

9. Conclusion

A new, fast and simple method has been introduced, which can be used to generate a pool of cipher keys and schedule keys. In this method there is no need for key expansion in both ciphering and deciphering process, the index of the schedule key and the key length can be included within the ciphered data block. This makes it difficult for an illegitimate receiver to guess the schedule key. This approach provides a huge number of probable schedule keys, which increases the security level and makes it almost impossible for anyone to predict the schedule key.

References

[1] Amasci Page, "Symmetric Ciphers," available at: http://www.amasci.com/~weidai/scan-mirror /cs.h tml, 2004.

[2] Aoki K. and Lipmaa H., “Fast Implementations of AES Candidates,” *in Proceedings of the 3rd AES Candidate Conference*, New York, USA, April 2000.

[3] Biryukov A. and Cannière C., “Block Ciphers and Systems of Quadratic Equations,” *in Proceedings of the Fast Software Encryption Workshop (FSE)*, pp. 274-289, 2003.

[4] Contini S., Rivest R. L., Robshaw M. J. B., and Yin Y. L., “Comments on the First Round AES Evaluation of RC6,” available at: http://csrc.nist.gov/encryption/aes/round1/pubcmnts.htm, 2000.

[5] Daemen J. and Rijmen V., “Answer to New Observations on Rijndael,” available at: http://csrc.nist.gov/aes, August 2000.

[6] Daemen J. and Rijmen V., “Rijndael/AES,” available at: www.ncipher.com/resources/downloads/files/datasheets/ciphertools.pdf, August 2003.

[7] Hellekalek P. and Wegenkittl, S., “Empirical Evidence Concerning AES,” *ACM Transactions on Modeling and Computer Simulation*, vol. 13, no. 4, pp. 322-333, October 2003.

[8] Lee Y. and Park Y., “Implementation of Rijndael Block Cipher Algorithm,” *in Proceedings of the 2002 International Technical Conference on Circuits/Systems*, Computers and Communic-ations, available at: http://www.kmutt.ac.th/itc 2002/Technical/final_program.html, July 2002.

[9] NIST 2001a, *Federal Information Processing Standards Publication (FIPS PUB) 197*, NIST, AES Page, available at: http://www.nist.gov/publications, 2004.

[10] Rashed A. and Ajlouni N., “An Extended Rijndael Block Cipher Using Java,” *in Proceedings of the 2004 International Conference on software Engineering Research and Practice*, Las Vigas, Nevada USA, June 2004.

[11] Rashed A., “Intelligent Encryption Decryption Systems,” *PhD Thesis*, Computer Information System Department, Arab Academy for Banking and Financial Sciences, 2004.

[12] Rosenthal J., “A Polynomial Description of the Rijndael, Advanced Encryption Standard,” available at: http://www.nd.edu/˜rosen/, February 2003.

[13] RSA Security Page, “Cryptography Encryption from RSA Security, What is Cryptography?,” available at: http://www.rsasecurity.com/rsalabs/faq/1-2.html, 2004.

[14] Search Security Page, “Rijndael,” available at: http://searchsecurity.techtarget.com/sDefinition/0sid14_gci523541,00.html, 2004.

[15] Signal Guard Page, “Key Management,” available at: http://www.signalguard.com/noframes/security/publickey.htm, 2004.

[16] Sklavos N. and Koufopavlou O., “Architectures and VLSI Implementations of the AES-Proposal Rijndael,” *IEEE Transactions on Computers*, vol. 51, no. 12, pp. 1454-1459, 2002.

Naim Ajlouni obtained his BSc in electricity and electronics engineering from Salford University in 1983, his MSc degree in robotics from Salford University in 1993, PhD in intelligent controlling from Salford University in 1997. Currently, he is an associate professor and the dean of the College of Graduate Computer Studies, Amman Arab University for Graduate Studies, Jordan. His research interests include genetic algorithms, fuzzy logic,neural networks , and security systems.

Asim El-Sheikh obtained his BSc (honours) in statistics from University of Kharoum in 1973, his MSc degree in operational research from University of London in 1983, and his PhD in computer simulation from University of London in 1987. Currently, he is an associate professor and the dean of the Faculty of Information Systems and Technology, Arab Academy for Banking and Financial Sciences, Jordan. His research interests include software engineering, software piracy, and ciphering.

Abdullah Abdali Rashed obtained his BSc in computer science from Applied Science University in 1997, Jordan, his MSc degree in information systems from Arab Academy for Banking and Financial Sciences in 2000, and his PhD in intelligent encryption decryption systems from Arab Academy for Banking and Financial Sciences in 2004. Currently, he is an assistant professor at Computer Information System Department, Arab Academy for Banking and Financial Sciences, Jordan. His research interests include block cipher cryptosystems, software piracy, genetic algorithms, intelligent systems, and computer languages.